Journal of Machine Learning Research 1 (2012) 1-48 Submitted 9/12; Published ?/??

Improving Markov Network Structure Learning Using Decision Trees

Daniel Lowd LOWD@CS.UOREGON.EDU
Department of Computer and Information Science
University of Oregon
Eugene, OR 97403, USA

Jesse Davis JESSE.DAVIS@CS.KULEUVEN.BE
Department of Computer Science
Katholieke Universiteit Leuven
3001 Heverlee, Belgium

Editor: ??

Abstract

Most existing algorithms for learning Markov network structure either are limited to learning interactions among few variables or are very slow, due to the large space of possible structures. In this paper, we propose three new methods for using decision trees to learn Markov network structures. The advantage of using decision trees is that they are very fast to learn and can represent complex interactions among many variables. The first method, DTSL, learns a decision tree to predict each variable and converts each tree into a set of conjunctive features that define the Markov network structure. The second, DT-BLM, builds on DTSL by using it to initialize a search-based Markov network learning algorithm recently proposed by Davis and Domingos (2010). The third, DT+L1, combines the features learned by DTSL with those learned by an L1-regularized logistic regression method (L1) proposed by Ravikumar et al. (2009). In an extensive empirical evaluation on 20 datasets, DTSL is comparable to L1 and significantly faster and more accurate than two other baselines. DT-BLM is slower than DTSL, but obtains slightly higher accuracy. DT+L1 combines the strengths of DTSL and L1 to perform significantly better than either of them with only a modest increase in training time.

Keywords: Markov networks, structure learning, decision trees, probabilistic methods

1. Introduction

A Markov network is an undirected, probabilistic graphical model for compactly representing a joint probability distribution over a set of random variables. In general, these variables can be discrete, continuous, or a mix; in this paper, we consider the case when all variables are discrete. Markov networks have been widely used in a number of domains, including computer vision, computational biology, and natural language processing. The structure of a Markov network defines which direct interactions among the variables are included in the model. This structure can be represented as a set of features, each of which is a Boolean-valued function of a subset of the variables. The parameters of a Markov network define the relative strength of those interactions. Selecting a Markov network structure that

includes the most important interactions in a domain is therefore essential for building an accurate model of that domain.

For some tasks, such as image processing, the structure of the Markov network may be hand crafted to fit the problem. In other problems, the structure is unknown and must be learned from data. These learned structures may be interesting in themselves, since they show the most significant direct interactions in the domain. In many domains, however, the goal is not an interpretable structure but an accurate final model. It is this last scenario that is the focus of our paper: learning the structure of a Markov network in order to accurately estimate marginal and conditional probabilities.

Learning an effective structure is difficult due to the very large structure space—the number of possible sets of conjunctive features is doubly-exponential in the number of variables. As a result, most previous approaches to learning Markov network structure are either very slow or limited to relatively simple features, such as only allowing pairwise interactions. In this paper, we propose to overcome these limitations by using decision tree learners, which are able to quickly learn complex structures involving many variables.

Our first method, DTSL (Decision Tree Structure Learner), learns probabilistic decision trees to predict the value of each variable and then converts the trees into sets of conjunctive features. We propose and evaluate several different methods for performing the conversion. Finally, DTSL merges all learned features into a global model. Weights for these features can be learned using any standard Markov network weight learning method. DTSL is similar in spirit to work by Ravikumar et al. (2010), who learn a sparse logistic regression model for each variable and combine the features from each local model into a global network structure. DTSL can also be viewed as converting a dependency network (Heckerman et al., 2000) with decision trees into a consistent Markov network.

Our second method, DT-BLM (Decision Tree Bottom-Up Learning), builds on DTSL by using the BLM algorithm of Davis and Domingos (2010) to further refine the structure learned by DTSL. This algorithm is much slower, but usually more accurate than DTSL. Furthermore, it serves as an example of how decision trees can be used to improve search-based structure learning algorithms by providing a good initial structure.

Our third method, DT+L1, combines the structure learned by DTSL with the pairwise interactions learned by L1-regularized logistic regression (L1) (Ravikumar et al., 2010). The trees used by DTSL are good at capturing higher-order interactions, but each leaf is mutually exclusive. In contrast, L1 captures many independent interaction terms, but each interaction is between just two variables. Their combination offers the potential to represent both kinds of interaction, leading to better performance in many domains.

We conducted an extensive empirical evaluation on 20 real-world datasets. We found that DTSL offers similar accuracy and speed as L1, performing better on datasets where it finds interesting tree structure and worse on datasets where it does not. Over 90% of the running time was spent learning weights, so there is potential to improve learning times even more with more sophisticated weight learning algorithms. The hybrid DT-BLM algorithm is often more accurate than DTSL, but is also much slower due to the additional refinement step. DT+L1 often has the best overall accuracy and runs much faster than DT-BLM, making it a very good algorithm overall. We also evaluated two other baseline structure learners, but they were not competitive with L1 and the three variants of DTSL.

This journal paper is an extended and improved version of the conference paper (Lowd and Davis, 2010). The extensions include two additional algorithms (DT-BLM and DT+L1) and more extensive experiments, including seven additional datasets and learning curves. The presentation has also been expanded and polished.

2. Markov Networks

This section provides a basic overview about Markov networks.

2.1 Representation

A *Markov network* is a model for the joint probability distribution of a set of variables $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ (Della Pietra et al., 1997). It is often expressed as an undirected graph G and a set of potential functions ϕ_k. The graph has a node for each variable, and the model has a potential function for each clique in the graph. The joint distribution represented by a Markov network is:

$$P(\mathbf{X}{=}\mathbf{x}) = \frac{1}{Z} \prod_k \phi_k(\mathbf{x}_{\{k\}}) \tag{1}$$

where $\mathbf{x}_{\{k\}}$ is the state of the variables that appear in the kth clique, and Z is a normalization constant called the *partition function*.

The graph encodes the following conditional independencies: sets of variables $\mathbf{X}_A$ and $\mathbf{X}_B$ are conditionally independent given evidence $\mathbf{Y}$ if all paths between their corresponding nodes in the graph pass through nodes from $\mathbf{Y}$. Any probability distribution that can be represented as a product of potential functions over the cliques of the graph, as in Equation (1), satisfies these independencies; for positive distributions, the converse holds as well.

One of the limitations of the graph structure is that it says nothing about the structure of the potential functions themselves. The most standard representation of a potential function over discrete variables is a table with one value for each variable configuration, but this requires a number of parameters that is exponential in the size of the clique. To learn an effective probability distribution, we typically need a finer-grained parametrization that permits a compact distribution even when the cliques are relatively large.

Therefore, we focus on learning the *log-linear representation* of a Markov network, in which the clique potentials are replaced by an exponentiated weighted sum of features of the state:

$$P(\mathbf{X}{=}\mathbf{x}) = \frac{1}{Z} \exp\left(\sum_j w_j f_j(\mathbf{x}_{\{j\}})\right) \tag{2}$$

A feature $f_j(\mathbf{x}_{\{j\}})$ may be any real-valued function of the state. For discrete data, a feature typically is a conjunction of tests of the form $X_i = x_i$, where X_i is a variable and x_i is a value of that variable. We say that a feature *matches* an example if it is true of that example. Any positive probability distribution over a discrete domain can be represented as log-linear model with conjunctive features. For example, a product of tabular potential functions could be converted into a log-linear model by constructing one conjunctive feature for each row of each table, using the log of the potential function value as the feature weight.

In this paper, we will refer to this set of conjunctive features as the structure of the Markov network. This detailed structure specifies not only the independencies of the distribution, but also the specific interaction terms that are most significant. If desired, the simpler undirected graph structure can be constructed from the features by adding an edge between each pair of nodes whose variables appear together in a feature.

2.2 Inference

The main inference task in graphical models is to compute the conditional probability of some variables (the query) given the values of some others (the evidence), by summing out the remaining variables. This problem is #P-complete. Thus, approximate inference techniques are required. One widely used method is Markov chain Monte Carlo (MCMC) (Gilks et al., 1996), and in particular Gibbs sampling, which proceeds by sampling each variable in turn given its *Markov blanket*, the variables it appears with in some potential or feature. These samples can be used to answer probabilistic queries by counting the number of samples that satisfy each query and dividing by the total number of samples. Under modest assumptions, the distribution represented by these samples will eventually converge to the true distribution. However, convergence may require a very large number of samples, and detecting convergence is difficult.

2.3 Weight Learning

The goal of weight learning is to select feature weights that maximize a given objective function. One of the most popular objective functions is the log-likelihood of the training data. In a Markov network, the negative log-likelihood is a convex function of the weights, and thus weight learning can be posed as a convex optimization problem. However, this optimization typically requires evaluating the log-likelihood and its gradient in each iteration. This is typically intractable to compute exactly due to the partition function. Furthermore, an approximation may work poorly: Kulesza and Pereira (2007) have shown that approximate inference can mislead weight learning algorithms.

A more computationally efficient alternative, widely used in areas such as spatial statistics, social network modeling, and language processing, is to optimize the pseudo-likelihood or pseudo-log-likelihood (PLL) instead (Besag, 1975). Pseudo-likelihood is the product of the conditional probabilities of each variable given its Markov blanket; pseudo-log-likelihood is the log of the pseudo-likelihood:

$$\log P_w^{\bullet}(\mathbf{X}=\mathbf{x}) = \sum_{j=1}^{V} \sum_{i=1}^{N} \log P_w(X_{i,j}=x_{i,j}|MB_x(X_{i,j})) \tag{3}$$

where V is the number of variables, N is the number of examples, $x_{i,j}$ is the value of the jth variable of the ith example, $MB_x(X_{i,j})$ is the state of $X_{i,j}$'s Markov blanket in the data. PLL and its gradient can be computed efficiently and optimized using any standard convex optimization algorithm, since the negative PLL of a Markov network is also convex.

3. Structure Learning in Markov Networks

Our goal in structure learning is to find a succinct set of features that can be used to accurately represent a probability distribution in a domain of interest. Other goals include learning the independencies or causal structure in the domain, but we focus on accurate probabilities. In this section, we briefly review four categories of approaches for Markov network structure learning, along with their strengths and weaknesses.

Global Search-Based Learning. One of the common approaches is to perform a global search for a set of features that accurately captures high-probability regions of the instance space (Della Pietra et al., 1997; McCallum, 2003). The algorithm of Della Pietra et al. (1997) is the most canonical example of this approach. The algorithm starts with a set of atomic features, each consisting of one state of one variable. It creates candidate features by conjoining each feature to each other feature, including the original atomic features. It calculates the weight for each candidate feature by assuming that all other feature weights remain unchanged, which is done for efficiency reasons. It uses Gibbs sampling for inference when setting the weight. Then, it evaluates each candidate feature f by estimating how much adding f would increase the log-likelihood. It adds the feature that results in the largest gain to the feature set. This procedure terminates when no candidate feature improves the model's score.

Recently, Davis and Domingos (2010) proposed an alternative bottom-up approach, called Bottom-up Learning of Markov Networks (BLM), for learning the structure of a Markov network. BLM starts by treating each complete example as a long feature in the Markov network. The algorithm repeatedly iterates through the feature set. It considers generalizing each feature to match its k nearest previously unmatched examples by dropping variables. If incorporating the newly generalized feature improves the model's score, it is retained in the model. The process terminates when no generalization improves the score.

These discrete search approaches are often slow due to the exponential number of possible features, leading to a doubly-exponential space of possible structures. Even a greedy search through this space must use the training data to repeatedly evaluate many candidates.

Optimization-Based Learning. Instead of performing a discrete search through possible structures, other recent work has framed the search as a continuous weight optimization problem with L1 regularization for sparsity (Lee et al., 2007; Schmidt and Murphy, 2010). The final structure consists of all features that are assigned non-zero weights. These methods are somewhat more efficient, but are typically limited to relatively short features. For example, in the approach of Lee et al. (2007), the set of candidate features must be specified in advance, and must be small enough that the gradient of all feature weights can be computed. Even including interactions among three variables requires a cubic number of features. Learning higher-order interactions quickly becomes infeasible. Schmidt and Murphy (2010) propose an algorithm that can learn longer features, as long as they satisfy a hierarchical constraint: longer features are only included when all subsets of the feature have been assigned non-zero weights. In experiments, this method does identify some longer features, but most features are short.

Independence Test Based Learning. Another line of work attempts to identify the Markov network structure directly by performing independence tests (Spirtes et al., 1993).

The basic idea is that if two variables are conditionally independent given some other variables then there should be no edge between them in the Markov network. Thus, instead of searching for interactions among the variables, these methods search for independencies. The challenge is the large number of conditional independencies to test: simply testing for marginal independence among each pair of variables is quadratic in the number of variables, and the complexity grows exponentially with the size of the separating set. Some variants of this approach search for the Markov blanket of each variable, the minimal set of variables that renders it conditionally independent from all others (Bromberg et al., 2009). Using independencies in the data to infer additional independencies can speed up this search, but many tests are still required. Furthermore, reliably recovering the independencies may not necessarily lead to the most accurate probabilistic model, since that is not the primary goal of these methods.

Learning Local Models. Ravikumar et al. (2010) proposed the alternative idea of learning a local model for each variable and then combining these models into a global model. Their method learns the structure by trying to discover the Markov blanket of each variable. It considers each variable X_i in turn and builds an L1-regularized logistic regression model to predict the value of X_i given the remaining variables. L1 regularization encourages sparsity, so that most of the variables end up with a weight of zero. The Markov blanket of X_i is all variables that have non-zero weight in the logistic regression model. Under certain conditions, this is a consistent estimator of the structure of a pairwise Markov network. In practice, when learned from real-world data, these Markov blankets are often incompatible with each other; for example, X_i may be in the inferred Markov blanket of X_j while the reverse does not hold. There are two methods for resolving these conflicts. One is to include an edge if either X_i is in X_j's Markov blanket or X_j is in X_i's Markov blanket. The other method is to include an edge only if X_i is in X_j's Markov blanket and X_j is in X_i's Markov blanket. In the final model, if there is an edge between X_i and X_j then the log-linear model includes a pairwise feature involving those two variables. All weights are then learned globally using any standard weight learning algorithm. While this approach greatly improves the tractability of structure learning, it is limited to modeling pairwise interactions, ignoring all higher-order effects. Furthermore, it still exhibits long run times for domains that have large numbers of variables.

4. Decision Tree Structure Learning (DTSL)

We now describe our method for learning Markov network structure from data, decision tree structure learning (DTSL). Algorithm 1 outlines our basic approach. For each variable X_i, we learn a probabilistic decision tree to represent the conditional probability of X_i given all other variables, $P(X_i|\mathbf{X} - X_i)$. Each tree is converted to a set of conjunctive features capable of representing the same probability distribution as the tree. Finally, all features are taken together in a single model and weights are learned globally using any standard weight learning algorithm.

This is similar in spirit to learning a dependency network (Heckerman et al., 2000): Both dependency networks (with tree distributions) and DTSL learn a probabilistic decision tree for each variable and combine the trees to form a probabilistic model. However, a dependency network may not represent a consistent probability distribution, and inference

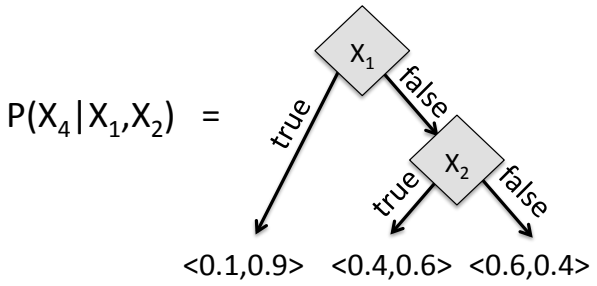

Figure 1: Example of a probabilistic decision tree.

can only be done by Gibbs sampling. In contrast, the Markov networks learned by DTSL always represent consistent probability distributions and allow inference to be done by any standard technique, such as loopy belief propagation (Murphy et al., 1999), mean field, or MCMC.

We now describe each step of DTSL in more detail.

Algorithm 1 The DTSL Algorithm

```
function DTSL(training examples D, variables X)
F ← ∅
for all X_i ∈ X do
  T_i ← LearnTree(D, X_i)
  F_i ← GenerateFeatures(T_i)
  F ← F ∪ F_i
end for
M ←LearnWeights(F, D)
return M
```

4.1 Learning Trees

A probabilistic decision tree represents a probability distribution over a target variable, X_i, given a set of inputs. Each interior node tests the value of an input variable and each of its outgoing edges is labeled with one of the outcomes of that test (e.g., true or false). Each leaf node contains the conditional distribution (e.g., multinomial) of the target variable given the test outcomes specified by its ancestor nodes and edges in the tree. We focus on discrete variables and consider tests of the form $X_j = x_j$, where X_j is a variable and x_j is value of that variable. Each conditional distribution is represented by a multinomial. Figure 1 contains an example of a probabilistic decision tree.

We can learn a probabilistic decision tree from data in a depth-first manner, one split at a time. We select a split at the root, partition the training data into the sets matching each outgoing branch, and recurse. We select each split to maximize the conditional log-likelihood of the target variable. This is very similar to using information gain as the split criterion. We used multinomials as the leaf distributions with a Dirichlet prior ($\alpha = 1$) for smoothing. In order to help avoid overfitting, we used a structure prior $P(S) \propto \kappa^p$,

where p is the number of parameters and $\kappa < 1$ represents a multiplicative penalty for each additional parameter in the model, as in Chickering et al. (1997). To further avoid overfitting, we set the minimum number of examples at each leaf to 10. Any splits that would result in fewer examples in a leaf are rejected.

Pseudocode for the tree learning subroutine is in Algorithm 2.

Algorithm 2 DTSL Tree Learning Subroutine

```
function LearnTree(training examples D, variable X_i)
best_split ← ∅
best_score ← 0
for all X_j ∈ X − X_i do
  for all x_j ∈ Val(X_j) do
    S ← (X_j = x_j)
    if Score(S, X_i, D) > best_score then
      best_split ← S
      best_score ←Score(S, X_i, D)
    end if
  end for
end for
if best_score > log κ then
  (D_t, D_f) ←SplitData(D, best_split)
  T_L ←LearnTree(D_t, X_i)
  T_R ←LearnTree(D_f, X_i)
  return new TreeVertex(best_split, T_L, T_R)
else
  Use D to estimate P(X_i)
  return new TreeLeaf(P(X_i))
end if
```

4.2 Generating Features

While decision trees are not commonly thought of as a log-linear model, any decision tree can be converted to a set of conjunctive features. In addition to a direct translation (Default), we explored four modifications (Prune, Prune-10, Prune-5, and Nonzero) which could yield structures with easier weight learning or better generalization.

The Default feature generation method is a direct translation of a probabilistic decision tree to an equivalent set of features. For each state of the target variable, we generate a feature for each path from the root to a leaf. The feature's conditions specify a single state of the target variable and all variable tests along the path in the decision tree. For example, to convert the decision tree in Figure 1 to a set of rules, we generate two features for each leaf, one where X_4 is true and one where X_4 is false. The complete list of features is as follows:

1. $X_1 = T \wedge X_4 = T$
2. $X_1 = T \wedge X_4 = F$

3. $X_1 = F \wedge X_2 = T \wedge X_4 = T$
4. $X_1 = F \wedge X_2 = T \wedge X_4 = F$
5. $X_1 = F \wedge X_2 = F \wedge X_4 = T$
6. $X_1 = F \wedge X_2 = F \wedge X_4 = F$

By using the log probability at the leaf as the rule's weight, we obtain a log linear model representing the same distribution. By applying this transformation to all decision trees, we obtain a set of conjunctive features that comprise the structure of our Markov network. However, their weights may be poorly calibrated (e.g., due to the same feature appearing in several decision trees), so weight learning is still necessary.

The Prune method expands the set of features generated by Default in order to make learning and inference easier. One disadvantage of the Default procedure is that it generates very long features with many conditions when the source trees are deep. Intuitively, we would like to capture the coarse interactions with short features and the finer interactions with longer features, rather than representing everything with long features. In the Prune method, we generate additional features for each path from the root to an interior node, not just paths from the root to a leaf. Each feature's conditions specify a single state of the target variable and all variable tests along the path in the decision tree. However, we include paths ending at any node, not just at leaves.

Specifically, for each state of the target variable and node in the tree (leaf or non-leaf), we generate a feature that specifies the state of the target variable and contains a condition for each ancestor of the node. This is equivalent to applying the Default feature generation method to all possible "pruned" versions of a decision tree, that is, where one or more interior nodes are replaced with leaves. This yields four additional rules, in addition to those enumerated above:

1. $X_4 = T$
2. $X_4 = F$
3. $X_1 = F \wedge X_4 = T$
4. $X_1 = F \wedge X_4 = F$

The Prune-10 and Prune-5 methods begin with the features generated by Prune and remove all features with more than 10 and 5 conditions, respectively. This can help avoid overfitting.

Our final feature generation method, Nonzero, is similar to Default, but removes all false variable constraints in a post-processing step. For example, the decision tree in Figure 1 would be converted to the following set of rules:

1. $X_1 = T \wedge X_4 = T$
2. $X_1 = T$
3. $X_2 = T \wedge X_4 = T$
4. $X_2 = T$
5. $X_4 = T$

This simplification is designed for sparse binary domains such as text, where a value of false or zero contains much less information than a value of true or one.

4.3 Asymptotic Complexity

Next, we explore DTSL's efficiency by analyzing its asymptotic complexity. Let n be the number of variables, m be the number of training examples, and l be the number of values per variable. The complexity of selecting the first split is $O(lmn)$, since we must compute statistics for each of the l values of each of the n variables using all of the m examples. At the next level, we now have two splits to select: one for the left child and one for the right child of the original split. However, since the split partitions the training data into two sets, each of the m examples is only considered once, either for the left split or the right split, leading to a total time of $O(lmn)$ at each level. If each split assigns a fraction of at least $1/k$ examples to each child, then the depth is at most $O(\log_k(m))$, yielding a total complexity of $O(lmn \log_k(m))$ for one tree, and $O(lmn^2 \log_k(m))$ for the entire structure. Depending on the patterns present in the data, the depth of the learned trees could be much less than $\log_k(m)$, leading to faster run times in practice. For large datasets or streaming data, we can apply the Hoeffding tree algorithm, which uses the Hoeffding bound to select decision tree splits after enough data has been seen to make a confident choice, rather than using all available data (Domingos and Hulten, 2000).

5. Decision Tree Bottom-Up Learning (DT-BLM)

The DTSL algorithm works by first using a decision tree learner to generate a set of conjunctive features and then learning weights for those features. In this section, we propose using decision trees within the context of the BLM Markov network structure learning algorithm (Davis and Domingos, 2010), which is described in Section 3.

BLM starts with a large set of long (i.e., specific) features and simplifies them to make them more general. The standard BLM algorithm uses the set of all training examples as the initial feature set. However, BLM could, in principle, generalize any set of initial features. The key idea of DT-BLM is to run BLM on the features from DTSL.

Algorithm 3 outlines the DT-BLM algorithm. DT-BLM receives a set of training examples, D, a set of variables, $\mathbf{X}$, and a set of integers, K, as input. It begins by running DTSL. Of the five feature conversion methods, it selects whichever one results in the best scoring model on validation data. Then DT-BLM employs the standard BLM learning procedure, but uses the features learned by DTSL as the initial feature set. The main loop in BLM involves repeatedly iterating through the feature set, calling the GeneralizeFeature method on each feature f in the feature set F.

The GeneralizeFeature method, outlined in Algorithm 4, proposes and scores several candidate feature generalizations. Specifically, it creates one generalization, f', for each $k \in K$ by finding the set of examples U_k which are f's k nearest unmatched examples. Next, it creates f' by dropping each variable-value test in f that does not match all examples in U_k, which has the effect of generalizing f' to match all examples in this set. It also scores the effect of removing f from F.

DT-BLM measures the distance between a feature and an example using the generalized value difference metric (GVDM), which tends to perform better in practice than the simpler

Hamming distance (Davis and Domingos, 2010). Formally, the distance, $D(f, e)$ is:

$$D(f, e) = \sum_{c \in f} GVDM(f, e, c) \tag{4}$$

where f is a feature, e is an example, c ranges over the variables in f and

$$GVDM(f, e, c) = \sum_{h} \sum_{f_i \in f, f_i \neq c} |P(c = h|f_i) - P(c = h|e_{f_i})|^Q$$

where h ranges over the possible values of variable c, f_i is the value of the ith variable in f, e_{f_i} is the value of the attribute referenced by f_i in e, and Q is an integer. For a variable c that appears in f, GVDM measures how well the other variables in f predict c. The intuition is that if c appears in a feature then the other variables should be good predictors of c.

Each generalization is evaluated by replacing f with the generalization in the model, relearning the weights of the modified feature set, and scoring the new model as:

$$S(D, F', \alpha) = PLL(D, F') - \alpha \sum_{f_i \in F'} |f_i| \tag{5}$$

where $PLL(D, F')$ is the train set pseudo-likelihood of feature set F', α is a penalty term to avoid overfitting, and $|f_i|$ is the number of variable-value tests in feature f_i. The procedure returns the best scoring generalized feature set, F'. DT-BLM updates F to F' if F' has a better score. The process terminates after making one full loop over the feature set without changing F by accepting a generalization.

The advantage of DT-BLM over DTSL is that it can refine individual features based on their global contribution to the pseudo-likelihood. This can lead to simpler models in terms of both the number and length of the features. One advantage of DT-BLM over BLM is that the features selected by DTSL are already very effective, so BLM is less likely to end up in a bad local optimum. A second advantage is that it removes the restriction that BLM can never learn a model that has more features than examples. This is valuable for domains which are most effectively modeled with a large number of short features (e.g., text). The principle disadvantage of DT-BLM is speed: it can be much slower than DTSL, since it is doing a secondary search over feature simplifications. We have also found that DT-BLM is sensitive to the Gaussian weight prior used during the BLM structure search, unlike the standard BLM algorithm. This means that DT-BLM requires more tuning time than BLM, as discussed more extensively in our empirical evaluation in Section 7.

Note that DT-BLM is similar in spirit to Bayesian network structure learning algorithms that combine independence-test and search-based learning techniques (Tsamardinos et al., 2006). These algorithms work in two phases. In the first step, they identify a superset of the edges that could be included in the network using independence tests. In the second step, a search through the space of possible structures is performed, but it is restricted to only consider including candidate edges identified in the first step. Typically, a greedy, general-to-specific search is employed. These algorithms differ from DT-BLM in three key ways: DT-BLM searches for features and not edges; DT-BLM uses decision trees to identify candidate features and not independence tests; and DT-BLM uses a specific-to-general search and not a general-to-specific search to refine the structure.

Algorithm 3 The DT-BLM Algorithm

function DT-BLM(training examples D, variables $\boldsymbol{X}$, K)
$F \leftarrow \emptyset$
for all $X_i \in \boldsymbol{X}$ **do**
  $T_i \leftarrow$ LearnTree(X_i, D)
  $F_i \leftarrow$ GenerateFeatures(T_i)
  $F \leftarrow F \cup F_i$
end for
repeat
  for all features $f \in F$ **do**
    $F' \leftarrow$ GeneralizeFeature(D, F, f, K)
    if Score$(TS, F') >$ Score(TS, F) **then**
      $F \leftarrow F'$
    end if
  end for
until no generalization improves the score
return M

Algorithm 4 Feature generalization subroutine used by DT-BLM.

function GeneralizeFeature(training examples D, feature set F, feature f, K)
$F_{best} = F$
for all $k \in K$ **do**
  $U_k = k$ nearest examples to f that do not match f
  $f' = f$ excluding each test in f that does not match all examples in U_k.
  $F' \leftarrow F$ with f replaced by f'
  if Score$(D, F') >$ Score(D, F_{best}) **then**
    $F_{best} \leftarrow F'$
  end if
end for
$F' = F$ without f
if Score$(D, F') >$ Score(D, F_{best}) **then**
  $F_{best} \leftarrow F'$
end if
return F_{best}

6. Combining DTSL and L1 (DT+L1)

In this section we propose DT+L1, a very simple way to combine DTSL and Ravikumar et al.'s L1 algorithm. DT+L1 works as follows. First, DTSL and L1 are run normally. That is, each method is run to completion (i.e., learning both the features and weights) to find the best model. Second, DT+L1 takes the union of the best DTSL feature set and the best L1 feature set. Third, DT+L1 learns the weights for the combined feature set and returns this as the final model.

The advantage of this approach is that it combines the strengths of both algorithms. DTSL excels at learning long features that capture complex interactions. Ravikumar et al.'s L1 approach only learns pairwise features, which occur less frequently in DTSL's learned models. One disadvantage is that DT+L1 will be more time intensive than either approach independently. DT+L1 involves performing parameter tuning to select the best model for both DTSL and L1 and then another run of weight learning, including parameter tuning, on the combined feature set. Another potential problem is that the combined model will have more features, which may lead to a more complex inference task.

7. Empirical Evaluation

We evaluate our algorithms on 20 real-world datasets. The goals of our experiments are three-fold. First, we want to determine how the different feature generation methods affect the performance of DTSL and DT-BLM (Section 7.3). Second, we want to compare the accuracy of DTSL, DT-BLM, and DT+L1 to each other as well as to several state-of-the-art Markov network structure learners: the algorithm of Della Pietra et al. (1997), which we refer to as DP; BLM (Davis and Domingos, 2010); and L1-regularized logistic regression (Ravikumar et al., 2010) (Section 7.4). Finally, we want to compare the running time of these learning algorithms, since this greatly affects their practical utility (Section 7.5).

7.1 Methodology

We used DTSL, DT-BLM, DT+L1, and each of the baselines to learn structures on 20 datasets.

DTSL was implemented in OCaml. For both BLM and DP, we used the publicly available code of Davis and Domingos (2010). Since DT-BLM is built on both DTSL and BLM, it used a combination of the DTSL and BLM code as well. For Ravikumar et al.'s approach, we tried both the OWL-QN (Andrew and Gao, 2007) and LIBLINEAR (Fan et al., 2008) software packages. Our initial experiments and evaluation were done using OWL-QN, but we later discovered that LIBLINEAR was much faster with nearly identical accuracy. Therefore, to be as fair as possible to L1, we report the running times for LIBLINEAR.

The output of each structure learning algorithm is a set of conjunctive features. To learn weights, we optimized the pseudo-likelihood of the data via the limited-memory BFGS algorithm (Liu and Nocedal, 1989) since optimizing the likelihood of the data is prohibitively expensive for the domains we consider.

Like Lee et al. (2007), we evaluated our algorithm using test set conditional marginal log-likelihood (CMLL). To make results from different datasets more comparable, we report normalized CMLL (NCMLL), which is CMLL divided by the number of variables in the domain. Calculating the NCMLL requires dividing the variables into a query set Q and an evidence set E. Then, for each test example we computed:

$$\text{NCMLL}(X = x) = \frac{1}{|X|} \sum_{i \in Q} \log P(X_i = x_i | E)$$

For each domain, we divided the variables into four disjoint sets. One set served as the query variables while the remaining three sets served as evidence. We repeated this procedure so

that each set served as the query variables once. We computed the conditional marginal probabilities using Gibbs sampling, as implemented in the open-source Libra toolkit.[1] For all domains, we ran 10 independent chains, each with 100 burn-in samples and followed by 1,000 samples for computing the probability. CMLL is related to PLL (Equation 3), since both measure the ability of the model to predict individual variables given evidence. However, we used approximately 75% of the variables as evidence when computing CMLL, while PLL always uses all-but-one variable as evidence.

We tuned all algorithms using separate validation sets, the same validation sets used by Haaren and Davis (2012). For DTSL, we selected the structure prior κ for each domain that maximized the total log-likelihood of all probabilistic decision trees on the validation set. The values of κ we used were powers of 10, ranging from 0.0001 to 1.0. When learning the weights for each feature generation method, we placed a Gaussian prior with mean 0 on each feature weight and then tuned the standard deviation to maximize PLL on the validation set, with values of 100, 10, 1, and 0.1. For comparisons to other algorithms, we selected the DTSL model with the best pseudo-likelihood on the validation set. We chose to use pseudo-likelihood for tuning instead of CMLL because it is much more efficient to compute.

For L1, on each dataset we tried the following values of the LIBLINEAR tuning parameter C: 0.001, 0.01, 0.05, 0.1, 0.5, 1 and 5.[2] We also tried both methods of making the Markov blankets consistent. These parameter settings allowed us to explore a variety of different models, ranging from those containing all pairwise interactions to those that were very sparse. We also tuned the weight prior as we did with DTSL. Tuning the standard deviation of the Gaussian weight prior allowed us to get better results than reported by Davis and Domingos (2010).

For BLM and DP, we kept the tuning settings used by Davis and Domingos (2010). We tried performing additional tuning of the weight prior for BLM, but it did not lead to improved results. For DT+L1, we combined the DTSL and L1 structures and relearned the final weights, including tuning the Gaussian weight prior on the validation set.

All of our code is available at `http://ix.cs.uoregon.edu/~lowd/dtsl` under a modified BSD license.

7.2 Datasets

For our experiments, we used the same set of 20 domains as Haaren and Davis (2012), 13 of which were previously used by Davis and Domingos (2010).[3] All variables are binary-valued. Basic statistics for all datasets are in Table 1, ordered by number of variables in the domain. "Density" refers to the fraction of non-zero entries. Below, we provide additional information about each dataset.

We used four clickstream prediction domains: KDDCup 2000, MSNBC, Anonymous MSWeb,[4] and Kosarek.[5] Each data point was a single session, with one binary-valued

1. Available from `http://libra.cs.uoregon.edu/`.
2. C is the inverse of the L1 regularization weight λ used by OWL-QN (Andrew and Gao, 2007). Larger C values almost always resulted in generating all pairwise features.
3. Publicly available at `http://alchemy.cs.washington.edu/papers/davis10a`
4. Available from the UCI machine learning repository (Blake and Merz, 2000).
5. `http://fimi.ua.ac.be/data/`

Dataset	# Train Ex.	# Tune Ex.	# Test Ex.	# Vars	Density
1. NLTCS	16,181	2,157	3,236	16	0.332
2. MSNBC	291,326	38,843	58,265	17	0.166
3. KDDCup 2000	180,092	19,907	34,955	64	0.008
4. Plants	17,412	2,321	3,482	69	0.180
5. Audio	15,000	2,000	3,000	100	0.199
6. Jester	9,000	1,000	4,116	100	0.608
7. Netflix	15,000	2,000	3,000	100	0.541
8. Accidents	12,758	1,700	2,551	111	0.291
9. Retail	22,041	2,938	4,408	135	0.024
10. Pumsb Star	12,262	1,635	2,452	163	0.270
11. DNA	1,600	400	1,186	180	0.253
12. Kosarek	33,375	4,450	6,675	190	0.020
13. MSWeb	29,441	3,270	5,000	294	0.010
14. Book	8,700	1,159	1,739	500	0.016
15. EachMovie	4,524	1,002	591	500	0.059
16. WebKB	2,803	558	838	839	0.064
17. Reuters-52	6,532	1,028	1,540	889	0.036
18. 20 Newsgroups	11,293	3,764	3,764	910	0.049
19. BBC	1,670	225	330	1,058	0.078
20. Ad	2,461	327	491	1,556	0.008

Table 1: Dataset characteristics.

variable for each page, area, or category of the site, indicating if it was visited during that session or not. For KDD Cup 2000 (Kohavi et al., 2000), we used the subset of Hulten and Domingos (2002), which consisted of 65 page categories. We dropped one category that was never visited in the training data. The MSNBC anonymous web data contains information about which top-level MSNBC pages were visited during a single session. The MSWeb anonymous web data contains visit data for 294 areas (Vroots) of the Microsoft web site, collected during one week in February 1998. Kosarek is clickstream data from a Hungarian online news portal.

Five of our domains were from recommender systems: Audio, Book, EachMovie, Jester and Netflix. The Audio dataset consists of information about how often a user listened to a particular artist.[6] The data was provided by the company Audioscrobbler before it was acquired by Last.fm. We focused on the 100 most listened-to artists. We used a random subset of the data and reduced the problem to "listened to" or "did not listen to." The Book Crossing (Book) dataset (Ziegler et al., 2005) consists of a user's rating of how much they liked a book. We considered the 500 most frequently rated books. We reduced the problem to "rated" or "not rated" and considered all people who rated at least two of these books. EachMovie[7] is a collaborative filtering dataset in which users rate movies they have seen. We focused on the 500 most-rated movies, and reduced each variable to "rated" or

6. `http://www-etud.iro.umontreal.ca/~bergstrj/audioscrobbler_data.html`
7. Provided by Compaq at `http://research.compaq.com/SRC/eachmovie/`; no longer available for download, as of October 2004.

"not rated". The Jester dataset (Goldberg et al., 2001) consists of users' real-valued ratings for 100 jokes. For Jester, we selected all users who had rated all 100 jokes, and reduced their preferences to "like" and "dislike" by thresholding the real-valued preference ratings at zero. Finally, we considered a random subset of the Netflix challenge data and focused on the 100 most frequently rated movies. We reduced the problem to "rated" or "not rated."

We used four text domains: 20 Newsgroups, Reuters-52, WebKB[8], and BBC[9]. For 20 Newsgroups, we only considered words that appeared in at least 200 documents. For Reuters, WebKB, and BBC, we only considered words that appeared in at least 50 documents. For all four datasets, we created one binary feature for each word. The text domains contained roughly a 50-50 train-test split, whereas all other domains used around 75% of the data for the training, 10% for tuning, and 15% for testing. Thus we split the test set of these domains to make the proportion of data devoted to each task more closely match the other domains used in the empirical evaluation.

The remaining seven datasets have no unifying theme. Plants consists of different plant types and locations where they are found.[4] We constructed one binary feature for each location, which is true if the plant is found there. DNA[10] is DNA sequences for primate splice-junctions; we used the binary-valued encoding provided. The National Long Term Care Survey (NLTCS) data consist of binary variables that measure an individual's ability to perform different daily living activities.[11] Pumsb Star contains census data for population and housing.[5] Accidents contains anonymized traffic incident data.[5] Retail is market basket data from a Belgian retail store.[5]

7.3 Feature Generation Methods

First, we compared the accuracy of DTSL with different feature generation methods: Default, Prune, Prune-10, Prune-5, and Nonzero. Table 2 lists the NCMLL of each method on each dataset. For each dataset, the method with the best NCMLL on the test set is in bold, and the method with the best PLL on the validation set is underlined. The results are shown graphically in the top half of Figure 2. The datasets are shown in the same order as in Table 1. Each bar represents the *negative* NCMLL of DTSL with one feature generation method on one dataset. Lower is better. To make the differences easier to see, we subtracted the negative NCMLL for Default from each bar, so that positive values (above the x-axis) are worse than Default and negative values (below the x-axis) are better than Default.

For DTSL, Prune is more accurate than Default on 15 datasets. Prune-10 rarely improved on the accuracy of Prune and Prune-5 often did worse. Nonzero was the most accurate method on seven datasets for DTSL. Overall, Prune did better on more datasets, but Nonzero worked especially well on Audio, Jester, and Netflix, three relatively dense collaborative filtering datasets. When we investigated these datasets further, we found that Default, Prune, and Prune-10 were overfitting, since they obtained better PLLs than Nonzero on the training data but worse PLLs on the validation data. Prune-5 was underfitting, obtaining worse PLLs than Nonzero on both training and validation

8. http://web.ist.utl.pt/~acardoso/datasets/
9. http://mlg.ucd.ie/datasets/bbc.html
10. http://www.cs.sfu.ca/~wangk/ucidata/dataset/DNA/
11. http://lib.stat.cmu.edu/datasets/

	DTSL					DT-BLM				
Dataset	Default	Nonzero	Prune	Prune-10	Prune-5	Default	Nonzero	Prune	Prune-10	Prune-5
NLTCS	-0.328	-0.326	-0.326	**-0.325**	-0.326	**-0.324**	-0.326	**-0.324**	**-0.324**	-0.325
MSNBC	**-0.336**	-0.344	**-0.336**	-0.340	-0.358	**-0.336**	-0.344	**-0.336**	-0.339	-0.357
KDDCup 2000	**-0.032**	-0.033	**-0.032**	**-0.032**	**-0.032**	**-0.032**	-0.033	**-0.032**	**-0.032**	**-0.032**
Plants	-0.146	-0.148	**-0.143**	-0.145	-0.153	-0.142	-0.148	**-0.141**	-0.144	-0.153
Audio	-0.380	**-0.375**	-0.379	-0.379	-0.384	-0.375	**-0.373**	-0.375	-0.375	-0.380
Jester	-0.512	**-0.505**	-0.511	-0.511	-0.514	-0.507	**-0.503**	-0.507	-0.507	-0.509
Netflix	-0.543	**-0.532**	-0.541	-0.541	-0.545	-0.540	**-0.532**	-0.539	-0.539	-0.542
Accidents	-0.150	**-0.147**	-0.150	-0.150	-0.169	-0.148	-0.146	-0.146	**-0.145**	-0.167
Retail	**-0.078**	-0.079	-0.079	**-0.078**	-0.079	**-0.078**	-0.079	**-0.078**	**-0.078**	-0.079
Pumsb Star	-0.104	**-0.098**	-0.101	-0.102	-0.108	-0.105	**-0.099**	-0.100	**-0.099**	-0.110
DNA	**-0.384**	-0.385	**-0.384**	**-0.384**	**-0.384**	-0.384	-0.385	**-0.383**	**-0.383**	-0.384
Kosarek	**-0.053**	**-0.053**	**-0.053**	**-0.053**	**-0.053**	**-0.053**	**-0.053**	**-0.053**	**-0.053**	**-0.053**
MSWeb	**-0.029**	-0.030	**-0.029**	-0.030	-0.030	**-0.029**	-0.030	**-0.029**	**-0.029**	-0.030
Book	**-0.069**	-0.070	**-0.069**	**-0.069**	**-0.069**	**-0.068**	-0.069	**-0.068**	**-0.068**	-0.069
EachMovie	-0.109	-0.104	**-0.102**	**-0.102**	-0.105	-0.101	-0.102	**-0.100**	**-0.100**	-0.102
WebKB	**-0.179**	**-0.179**	**-0.179**	**-0.179**	-0.180	-0.178	-0.177	**-0.177**	**-0.177**	-0.178
Reuters-52	**-0.092**	**-0.092**	**-0.092**	-0.094	-0.093	-0.093	-0.092	**-0.091**	**-0.091**	-0.092
20 Newsgroups	-0.170	-0.169	**-0.166**	**-0.166**	-0.167	**-0.163**	-0.165	-0.165	-0.165	-0.165
BBC	-0.238	**-0.237**	-0.240	-0.240	-0.241	**-0.237**	**-0.237**	**-0.237**	**-0.237**	**-0.237**
Ad	**-0.012**	-0.144	-0.016	-0.016	-0.017	**-0.016**	-0.151	-0.020	-0.019	-0.022

Table 2: NCMLL of DTSL (left) and DT-BLM (right) with different conversion methods. For each algorithm and dataset, the method with the best test set NCMLL is in bold and the method with the best validation set PLL is underlined.

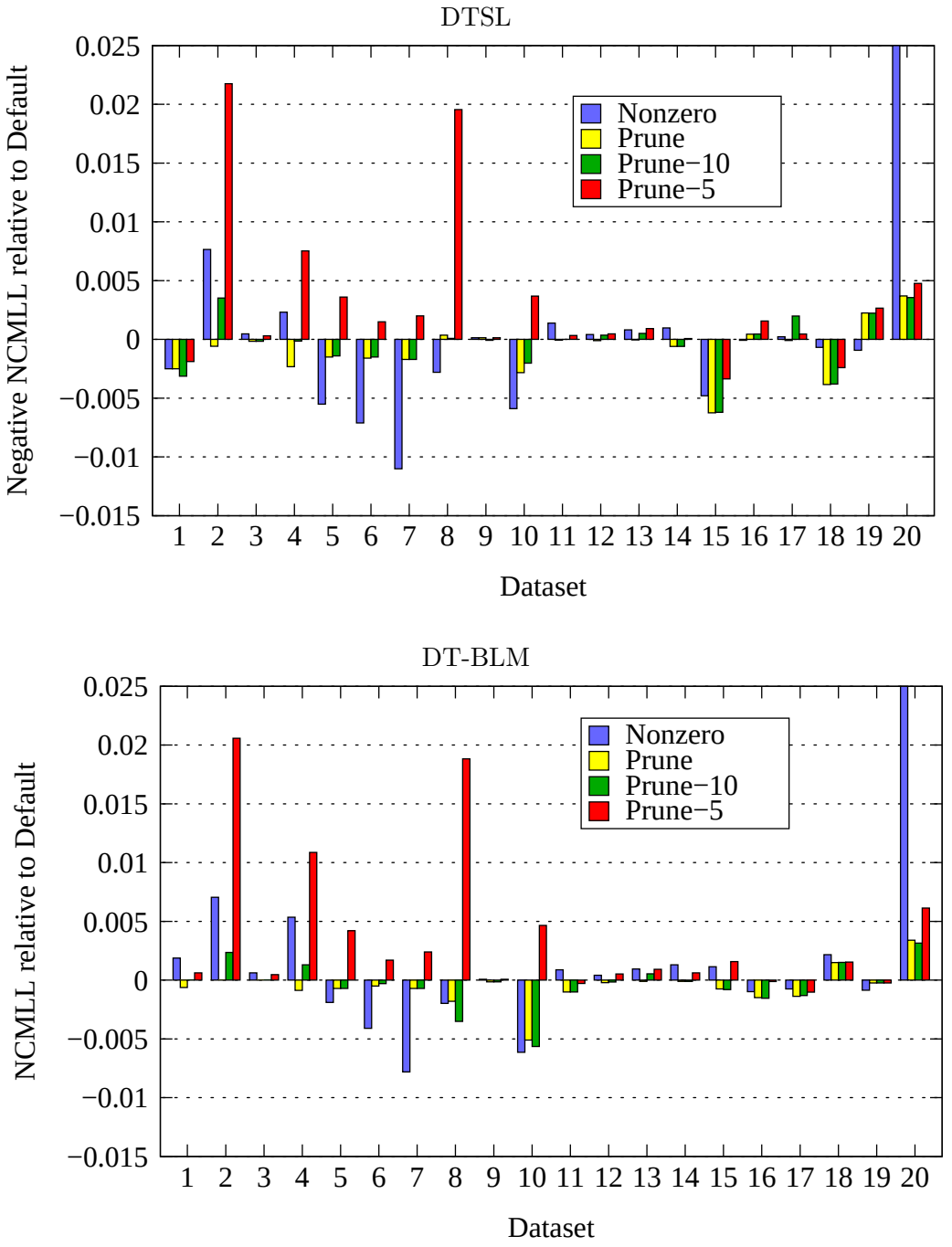

Figure 2: Performance of different feature conversion methods with DTSL (top) and DT-BLM (bottom), relative to Default. Lower values indicate better performance. Positive values (above the x-axis) indicate methods that performed worse than Default, and negative values (below the x-axis) indicate methods that performed better than Default.

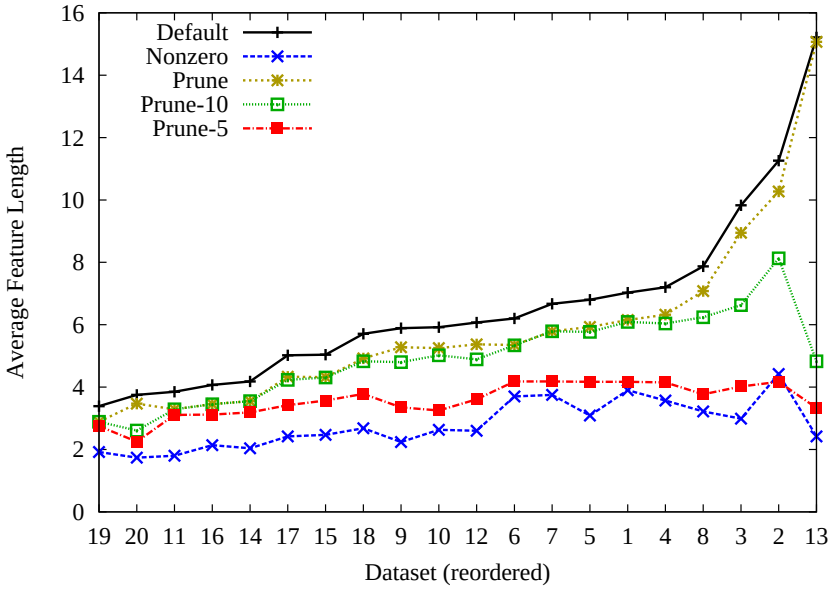

Figure 3: Average feature length for each DTSL feature generation method on each dataset. Datasets are ordered by the average feature length generated by the Default method.

data. We hypothesize that Nonzero provides beneficial regularization by removing many features. Long features are more likely to have one or more false variable constraints, and are therefore more likely to be removed by Nonzero. If these longer features are the source of the overfitting problems, then placing a stricter prior on the weights of longer features might offer a similar benefit.

The results on DT-BLM are similar, as shown in the right side of Table 2 and the bottom half of Figure 2. For DT-BLM, Prune is more accurate on 18 datasets, although many of these differences are very small. Nonzero is most accurate on five datasets, but Prune is relatively close on three of them. On average, the additional feature refinement done by DT-BLM seems to render it somewhat less sensitive to the choice of feature generation method.

Our tuning procedure uses the PLL of the validation set for model selection. Thus, the model we select may be different from the one with the best NCMLL on the test set, since it is selected according to a different metric on different evaluation data. For both DTSL and DT-BLM, the method selected with the validation set (underlined in Table 2) is often the same as the one with the best NCMLL (bold in Table 2). When they are different, the NCMLL of the alternative model is very close. This suggests that PLL does a reasonably good job of model selection for DTSL and DT-BLM on these datasets.

For DTSL, additional characteristics of the features generated by each method are shown in Figures 3 and 4. In each graph, the datasets have been sorted by the value of the Default method to make the comparison between methods clearer. "Average feature length" is the average number of conditions per feature. The Prune method leads to roughly twice as many features as Default, which is what one would expect, since half of the nodes in a

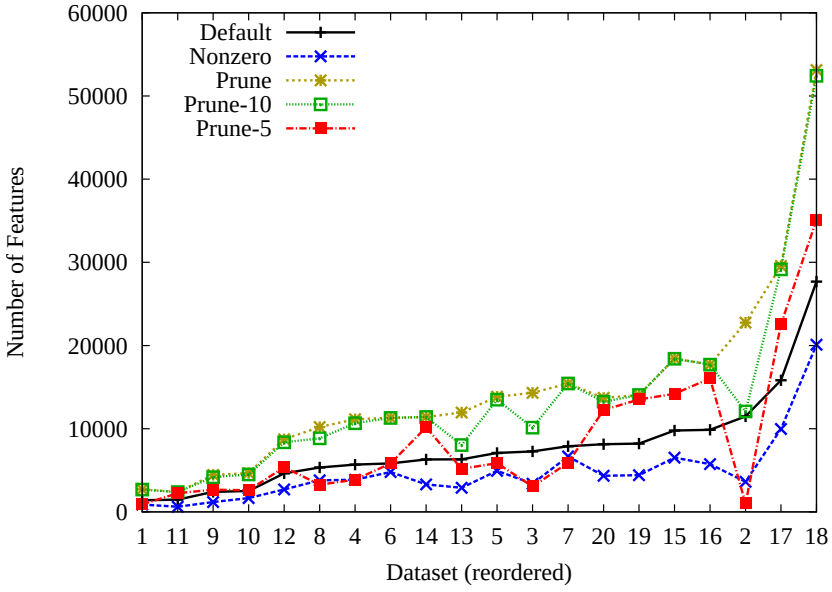

Figure 4: Number of features for each DTSL feature generation method on each dataset. Datasets are ordered by the number of features generated by the Default method.

balanced binary tree are leaves and the other half are interior nodes. Nonzero typically yields the shortest and the fewest rules, as expected.

7.4 Accuracy

We then compared DTSL, DT-BLM, and DT+L1 to three standard Markov network structure learners: L1-regularized logistic regression (Ravikumar et al., 2010), BLM (Davis and Domingos, 2010), and DP (Della Pietra et al., 1997). For DTSL and DT-BLM, we used the feature generation method that performed best on the validation set. In some cases, such as KDDCup 2000, this was not the method that performed best on the test data.

Figure 5 shows how DTSL, DT-BLM, DT+L1, and the three baselines compare in terms of NCMLL. For each dataset, bars above the x-axis indicate algorithms that perform worse than DTSL, and bars below the x-axis indicate algorithms that perform better. Raw numbers for NCMLL and NPLL can be found in Table 3. NPLL is the pseudo-log-likelihood divided by the number of variables in the domain. The NPLL results are qualitatively similar to the NCMLL results, except for Pumsb Star and DNA, where L1 ranks better according to NCMLL than NPLL, and 20 Newsgroups and BBC, where BLM ranks worse according to NCMLL than NPLL. Table 3 also shows which model has the best validation set NPLL. Note that in 19 out of the 20 datasets this corresponds to the model with the best NCMLL, indicating that PLL is a good objective function to optimize for this evaluation metric. We focus our subsequent discussion on the NCMLL results, which we believe to be a better measure of model accuracy than NPLL for typical queries.

	Test set CMLL						Test set NPLL					
Dataset	DP	BLM	L1	DTSL	DT-BLM	DT+L1	DP	BLM	L1	DTSL	DT-BLM	DT+L1
NLTCS	-0.326	-0.328	-0.327	-0.325	**-0.324**	-0.325	-0.307	-0.311	-0.309	-0.309	**-0.307**	-0.308
MSNBC	-0.349	-0.344	-0.369	-0.337	**-0.336**	**-0.336**	-0.299	-0.288	-0.356	**-0.252**	**-0.252**	**-0.252**
KDDCup 2000	-0.033	**-0.032**	-0.033	**-0.032**	**-0.032**	**-0.032**	-0.034	-0.032	-0.032	-0.032	**-0.031**	**-0.031**
Plants	-0.159	-0.151	-0.156	-0.143	**-0.141**	**-0.141**	-0.135	-0.133	-0.136	-0.124	**-0.122**	**-0.122**
Audio	-0.392	-0.375	**-0.370**	-0.375	-0.373	**-0.370**	-0.385	-0.368	**-0.362**	-0.370	-0.367	-0.366
Jester	-0.537	-0.530	**-0.496**	-0.505	-0.503	-0.504	-0.528	-0.526	**-0.488**	-0.498	-0.497	-0.492
Netflix	-0.574	-0.565	**-0.523**	-0.532	-0.532	-0.525	-0.561	-0.560	**-0.511**	-0.523	-0.524	-0.514
Accidents	-0.270	-0.339	-0.149	-0.147	-0.146	**-0.141**	-0.240	-0.296	-0.112	**-0.105**	-0.106	**-0.105**
Retail	-0.079	-0.079	-0.079	**-0.078**	**-0.078**	**-0.078**	-0.075	-0.077	**-0.076**	**-0.076**	**-0.076**	**-0.076**
Pumsb Star	-0.183	-0.623	-0.085	-0.101	-0.100	**-0.084**	-0.145	-0.176	**-0.059**	**-0.059**	**-0.059**	**-0.059**
DNA	-0.541	-0.554	**-0.384**	**-0.384**	**-0.384**	**-0.384**	-0.523	-0.545	-0.326	**-0.323**	**-0.323**	**-0.323**
Kosarek	-0.057	-0.054	-0.054	-0.053	-0.053	**-0.052**	-0.056	-0.053	-0.052	-0.052	**-0.051**	**-0.051**
MSWeb	-0.031	-0.030	-0.030	**-0.029**	**-0.029**	**-0.029**	-0.030	-0.029	-0.030	**-0.027**	**-0.027**	**-0.027**
Book	-0.078	-0.069	-0.073	-0.069	**-0.068**	**-0.068**	-0.076	-0.068	-0.073	-0.068	**-0.067**	**-0.067**
EachMovie	-0.134	-0.117	-0.104	-0.102	**-0.100**	**-0.100**	-0.132	-0.116	-0.101	-0.102	**-0.099**	**-0.099**
WebKB	-0.210	-0.196	-0.179	-0.179	-0.177	**-0.176**	-0.201	-0.193	-0.175	-0.175	-0.173	**-0.172**
Reuters-52	-0.119	-0.102	**-0.091**	-0.092	-0.092	**-0.091**	-0.130	-0.101	-0.090	-0.091	**-0.089**	-0.090
20 Newsgroups	-0.188	-0.176	**-0.165**	-0.169	**-0.165**	-0.166	-0.172	-0.175	**-0.163**	-0.167	**-0.163**	-0.166
BBC	-0.258	-0.251	-0.245	**-0.237**	**-0.237**	-0.240	-0.250	-0.247	-0.246	-0.236	**-0.235**	-0.241
Ad	-0.033	-0.025	**-0.006**	-0.017	-0.022	**-0.006**	-0.033	-0.008	**-0.004**	-0.008	-0.008	**-0.004**

Table 3: Test set NCMLL and NPLL for all algorithms. The DTSL feature generation method was selected using the validation set. The best result for each metric is shown in bold. The method with the best validation set PLL is underlined.

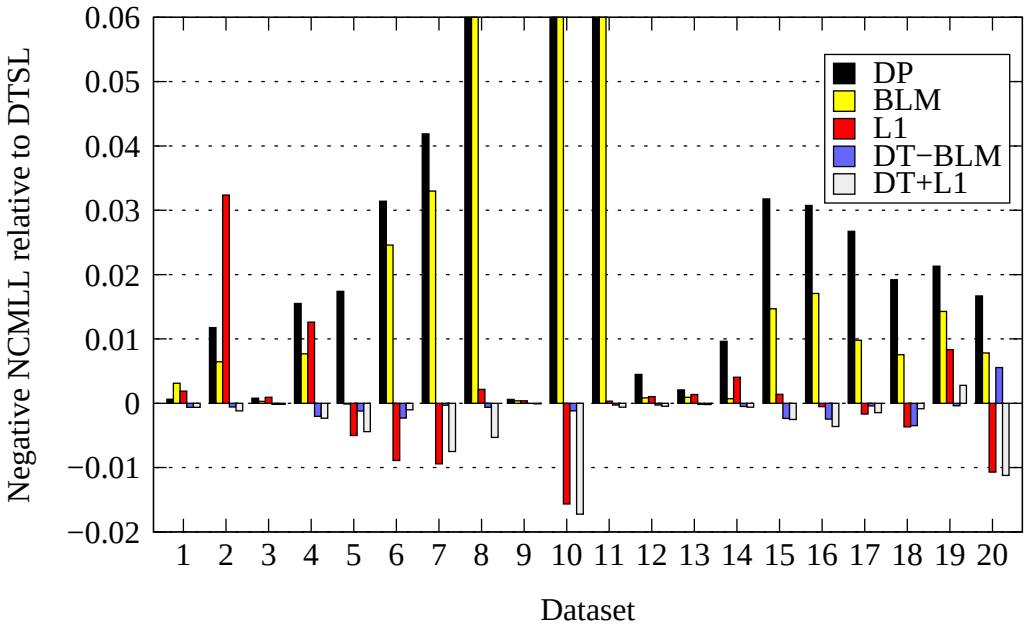

Figure 5: Normalized negative CMLL, relative to DTSL. Lower values indicate better performance. Positive values (above the x-axis) indicate methods that performed worse than DTSL, and negative values (below the x-axis) indicate methods that performed better than DTSL.

Overall, DP and BLM are fairly inaccurate, DTSL and L1 are roughly comparable, and DT-BLM is slightly better than DTSL. DT+L1 usually does at least as well as both DTSL and L1, making it the most reliably accurate algorithm overall. DTSL is always more accurate than DP and BLM, except for three datasets where it is tied with BLM. DTSL is significantly more accurate than both DP and BLM ($p < 0.001$) according to a Wilcoxon signed-ranks test in which the test set NCMLL of each dataset appears as one sample in the significance test. DT-BLM represents a modest improvement in accuracy over DTSL, performing slightly better than DTSL on 11 datasets and worse on only one. On the remaining eight datasets, the difference in NCMLL was less than 0.001. A Wilcoxon signed-ranks test indicates that DT-BLM is significantly more accurate than DTSL ($p < 0.05$). Since DT+L1 includes the features from both DTSL and L1, it usually does at least as well as both methods, and sometimes better. DT+L1 is the most accurate method (including ties) on 15 out of 20 datasets, while DT-BLM is one of the most accurate methods on only 11 datasets, DTSL on five, and L1 on seven. DT+L1 is more accurate than both L1 and DTSL ($p < 0.05$) according to a Wilcoxon signed-ranks test.

Comparisons between DTSL or DT-BLM and L1 are interesting because they demonstrate the relative strengths of using trees versus logistic regression for generating features. DTSL performs better than L1 on 11 datasets and worse on seven. Similarly, DT-BLM performs better than L1 on 12 datasets and worse on six. These differences are not significant according to a Wilcoxon signed-ranks test. The relative performance of DTSL and

L1 seems to vary greatly from dataset to dataset. To better understand what makes DTSL perform better or worse than L1, we examined the average length of the features learned by DTSL. For the domains where DTSL performs worse than L1, the average feature length is 3.22, indicating relatively shallow trees with simple features. For the domains where DTSL performs better, the average feature length is 6.04. This supports the hypothesis that DTSL does better on domains with higher-order interactions that can be discovered by decision trees, while L1 does better on domains with many low-order interactions that can be modeled as pairwise features. DT-BLM's features show a similar trend. Figure 6 shows the average feature length for each algorithm on each dataset, ordered by the average feature length with DTSL.

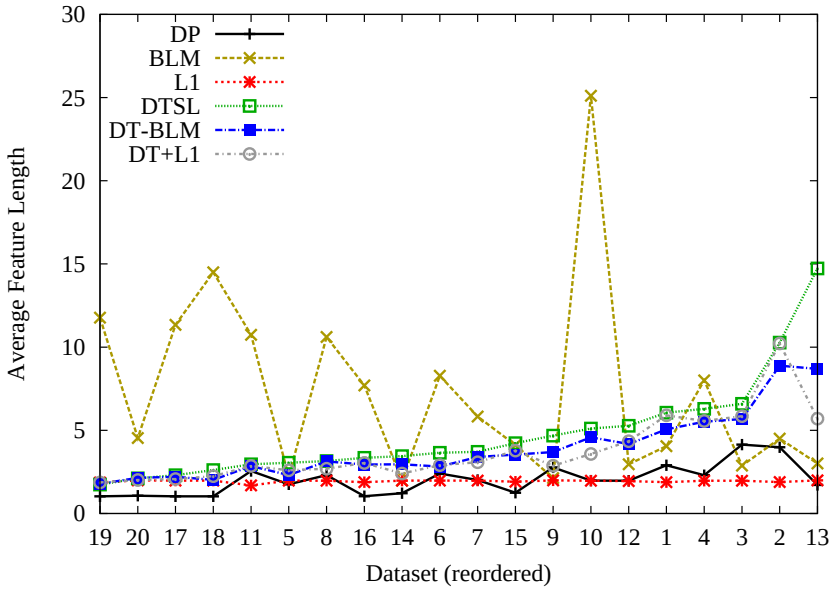

Figure 6: Average feature length for each algorithm on each dataset. Datasets are ordered by the average feature length with DTSL.

We also examined the number of features learned by each algorithm (see Figure 7). When DTSL learned many more features than L1, it typically did better than L1 (NLTCS, MSNBC, KDDCup, Plants, Kosarek, EachMovie) or the same (DNA, WebKB). However, when L1 learned many more features than DTSL, it sometimes did better (Reuters-52, Ad) and sometimes did worse (Retail, MSWeb, Book). Thus, the number of features learned does not appear to correlate strongly with the relative performance of these two algorithms.

Figures 8 and 9 contain learning curves comparing DTSL and DT-BLM to L1. For the most part, all three algorithms exhibit a similar dependence on the amount of training data. The exceptions to this are Pumsb Star and Ad, where DTSL shows signs of overfitting. The relative performance of L1 goes up and down somewhat in NLTCS, DNA, EachMovie, and 20 Newsgroups, but in most datasets the ranking of the methods is stable across all amounts of training data. For some datasets (MSNBC, KDDCup 2000, Plants), L1 appears to have converged to a different asymptotic error rate than DTSL, due to its different learning

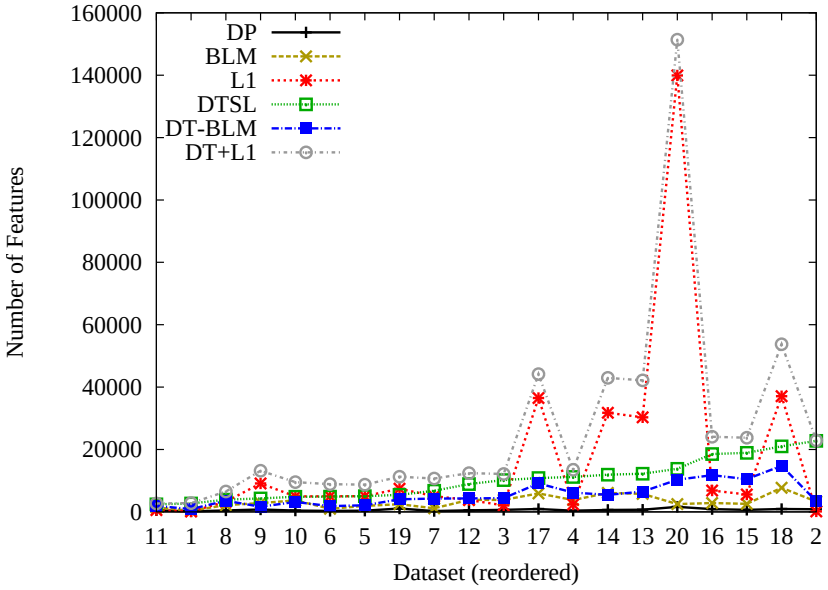

Figure 7: Number of features for each algorithm on each dataset. Datasets are ordered by the number of features generated by DTSL.

bias. This is consistent with the hypothesis that some datasets are simply better suited to the learning bias of L1, and some are better suited to the longer conjunctive features representable by trees.

7.5 Learning Time

A comparison of running times is shown in Figure 10 (excluding DP), with raw numbers in Table 4. The timing results shown include parameter tuning. L1 was fastest on nine datasets; DTSL was fastest on 10; and BLM was fastest on one dataset (Ad). Results excluding tuning time are similar – L1 is fastest on six datasets; DTSL on 13; and BLM on one. For DT+L1, we report the total time required for learning both the DTSL and L1 structures, as well as the additional time required for weight learning.

We use the geometric mean running time of each algorithm to summarize performance over all datasets. On average, DTSL is 5% slower than L1, 4.6 times faster than BLM, and 21.7 times faster than DP. DTSL is significantly faster than both BLM and DP according to a Wilcoxon signed-ranks test on the log of the training time ($p < 0.05$). Although the geometric mean running time of DTSL is slightly worse than L1's, its arithmetic mean is slightly better, mainly because DTSL tends to be faster on the larger, slower datasets such as 20 Newsgroups and BBC.

DT-BLM is 19.7 times slower than DTSL and 10.6 times slower than BLM; these differences are also significant under the same test ($p < 0.01$). DT-BLM is slower than BLM for two reasons. First, some datasets have more DTSL features than examples in the original training data. For example, BBC has only 1,670 examples but results in 5,475 features in DTSL's model. Since DT-BLM runs on DTSL features instead of the original training

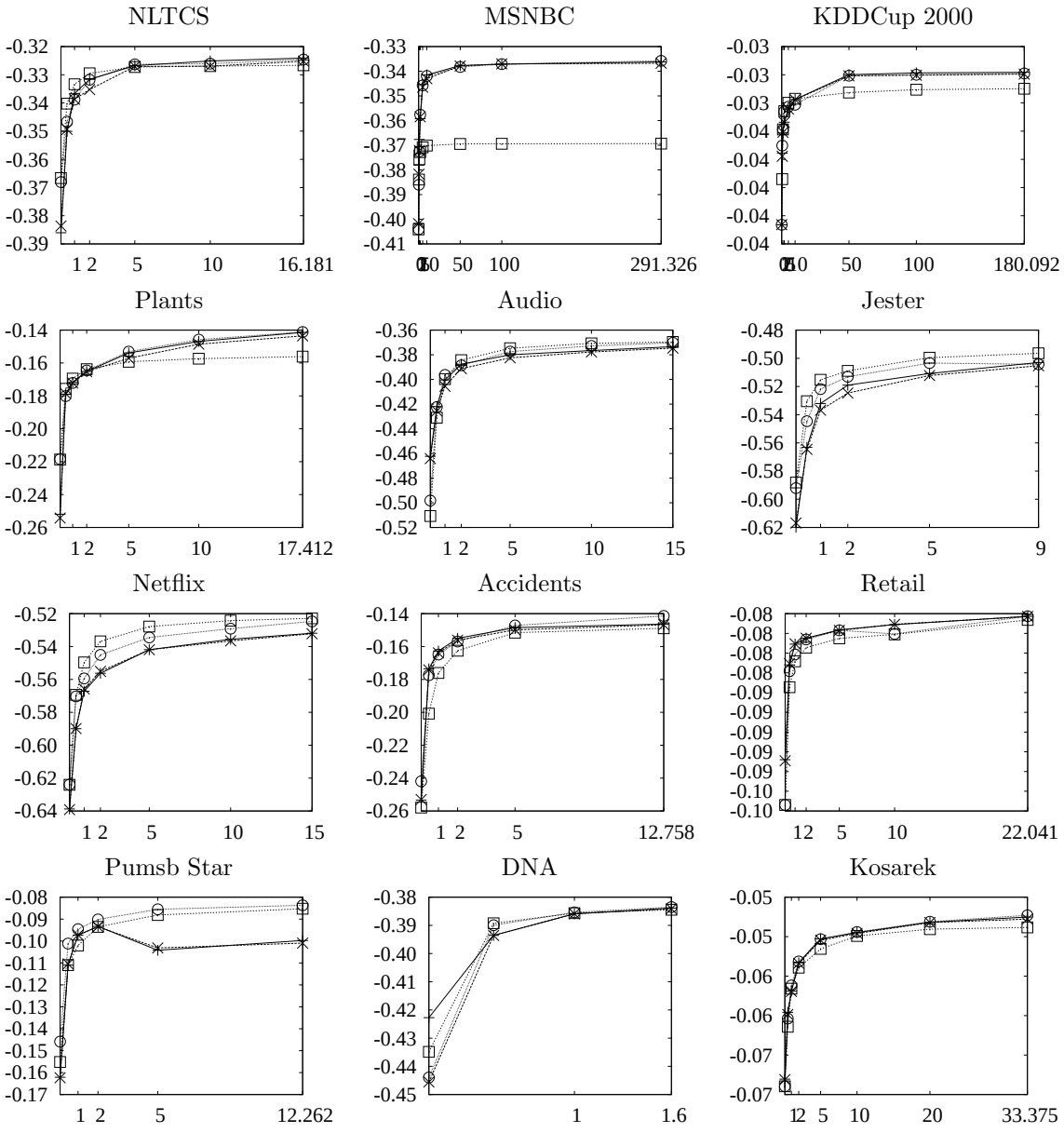

Figure 8: NCMLL vs. thousands of training examples for DT+L1 (circles), DT-BLM (pluses), DTSL (x marks), and L1 (boxes) on the first 12 datasets. Higher is better.

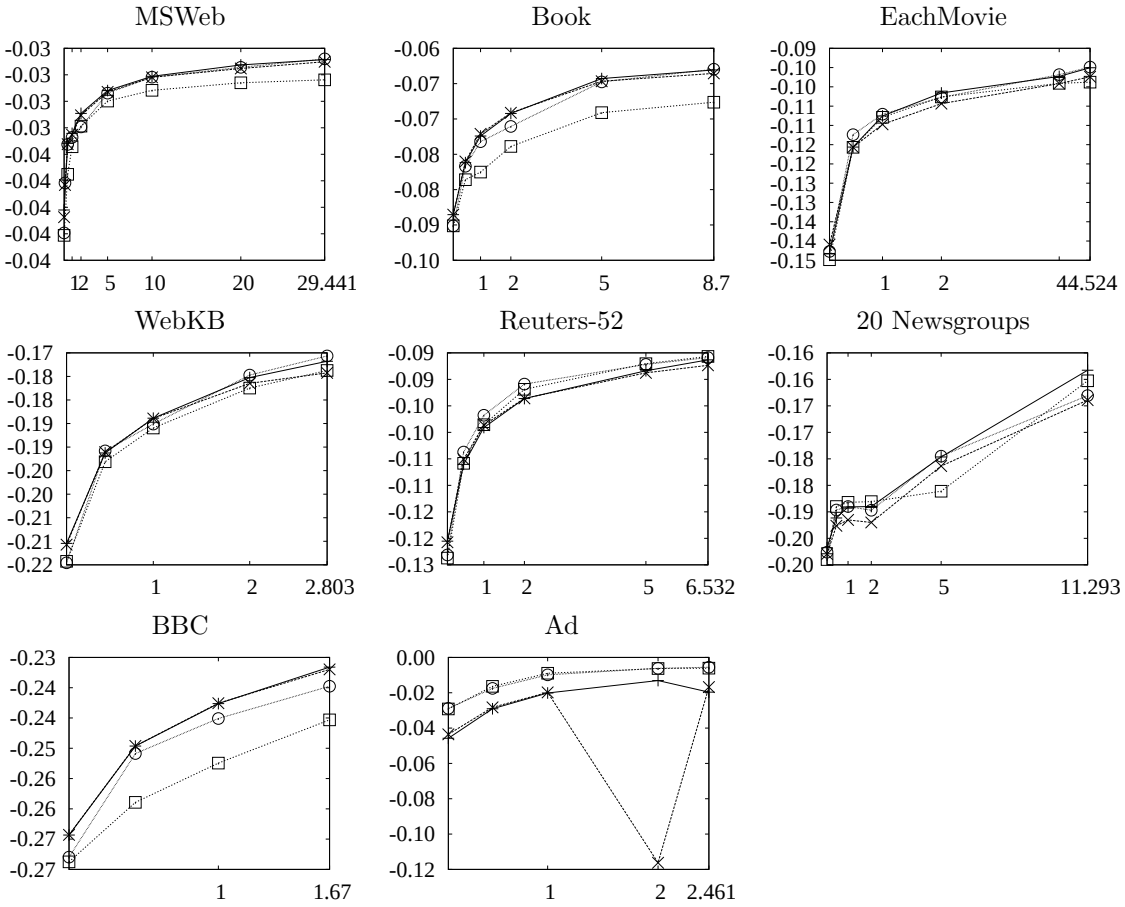

Figure 9: NCMLL vs. thousands of training examples for DT+L1 (circles), DT-BLM (pluses), DTSL (x marks), and L1 (boxes) on the last 8 datasets. Higher is better.

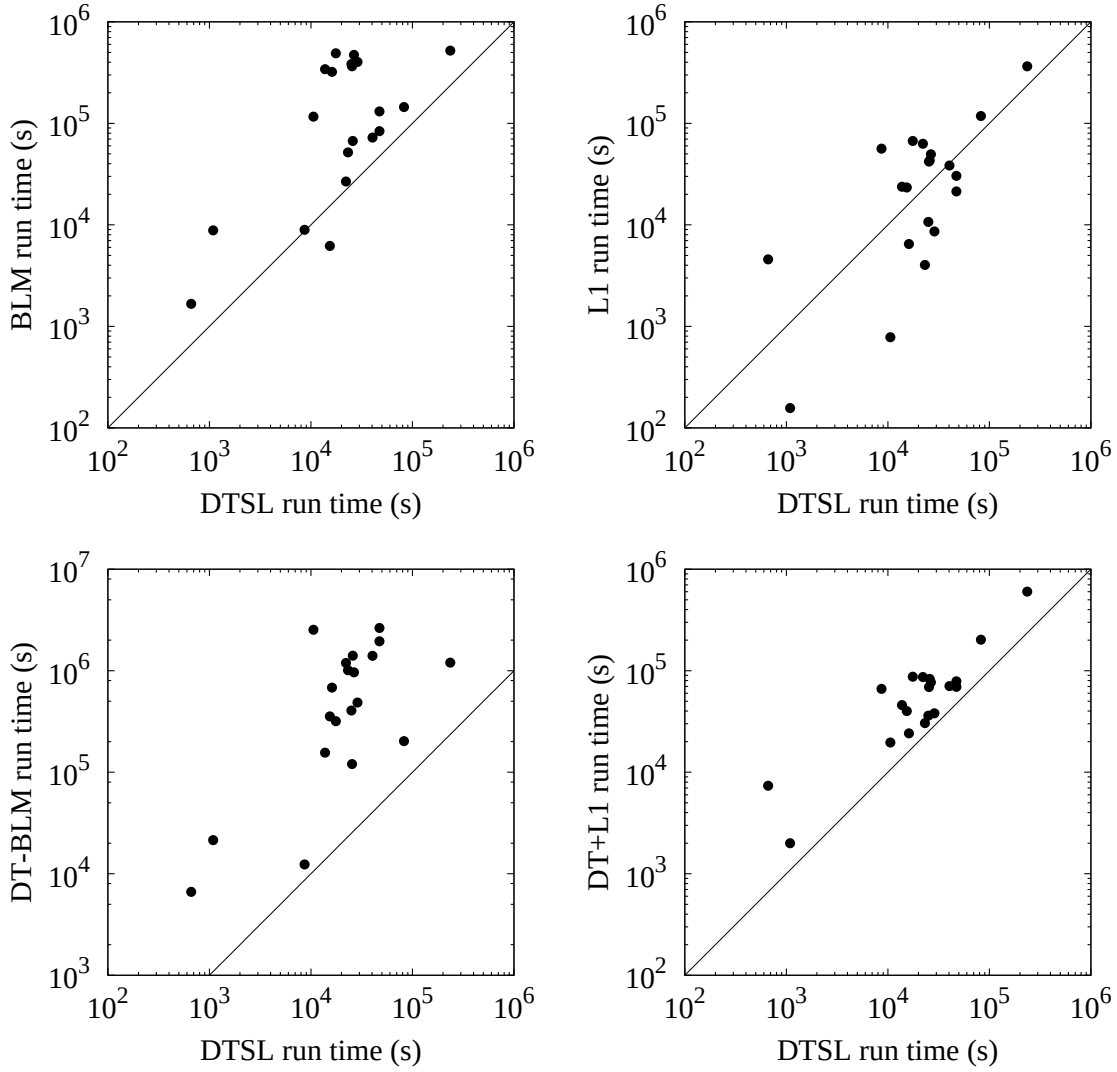

Figure 10: Running time on each dataset in seconds (y-axis) relative to DTSL's running time (x-axis). Points below the line y=x represent datasets where DTSL was slower than the other algorithm. All times include tuning.

data, more features means a longer running time. Second, DT-BLM is more sensitive to the width of the Gaussian prior than BLM is, so DT-BLM has one extra parameter to tune. If we instead use the best Gaussian prior width from DTSL, the algorithm runs roughly four times faster, but remains slower than BLM.

Table 5 shows a division of the total learning time, including tuning, divided into the time spent learning the structure (i.e., the features) and the time spent on weight learning. With respect to the number of variables, DTSL has better scaling characteristics for feature generation than Ravikumar et al.'s L1 approach. The number of variables seems to have a

Dataset	DP	BLM	L1	DTSL	DT-BLM	DT+L1
NLTCS	1,934	8,836	**157**	1,089	21,430	2,001
MSNBC	438,611	116,315	**782**	10,597	2,533,367	19,606
KDDCup 2000	335,398	51,744	**4,036**	23,201	1,012,635	30,463
Plants	423,593	321,880	**6,474**	16,118	682,405	24,145
Audio	653,111	402,441	**8,611**	28,738	155,970	45,829
Jester	583,969	341,830	23,769	**13,771**	485,892	38,078
Netflix	653,099	472,792	49,582	**26,560**	962,730	76,944
Accidents	637,574	364,282	41,982	**25,436**	120,434	69,045
Retail	279,507	384,011	**10,698**	25,030	405,059	36,090
Pumsb Star	707,530	489,854	66,971	**17,578**	318,017	87,292
DNA	197,406	1,666	4,570	**661**	6,635	7,365
Kosarek	626,336	130,962	**21,353**	47,363	1,947,259	69,363
MSWeb	426,199	83,727	**30,393**	47,363	2,635,238	78,579
Book	641,338	72,398	**38,444**	40,409	1,405,667	82,966
EachMovie	694,509	66,983	42,923	**25,873**	1,401,549	70,613
WebKB	715,885	26,754	63,010	**22,113**	1,192,672	86,727
Reuters-52	790,165	144,641	118,166	**82,467**	202,551	202,237
20 Newsgroups	792,268	520,361	364,512	**235,778**	1,201,506	601,137
BBC	864,000	8,944	56,293	**8,661**	12,352	66,197
Ad	719,893	**6,196**	23,375	15,373	354,843	40,083
Arith. Mean	559,116	200,831	48,805	**35,709**	852,911	86,738
Geom. Mean	416,573	87,876	**18,268**	19,172	378,405	48,662

Table 4: Run time in seconds, including parameter tuning. The best run time is shown in bold.

greater effect on run time than the number of examples. Each additional variable requires learning one extra model. Furthermore, each individual learning task is more complex because the target variable can depend on one additional input variable. The most striking observation is that the majority of time is spent learning the feature weights. For L1, on average, weight learning accounts for 93.8% of the total run time. For DTSL, this rises to 99.1% of the total run time. The factor that most influences weight learning time is the number of features: models with more features lead to longer weight learning times. Thus, more efficient weight learning techniques would substantially improve the running time of both structure learning algorithms.

7.6 Discussion

Both DTSL and L1 are typically faster and more accurate than DP and BLM. DTSL excels in domains that depend on higher-order interactions, while L1 performs better in domains that require many pairwise interactions. Therefore, neither algorithm dominates or subsumes the other; rather, they discover complementary types of structure.

DTSL has two weaknesses. The first is a higher risk of overfitting, since it often generates many very specialized features. For the most part, this can be remedied with careful tuning on a validation set. The second is a limited ability to capture many independent interactions. For instance, to capture pairwise interactions between a variable and k other variables would

	L1 Learning Times			DTSL Learning Times		
Dataset	Structure	Weights	Total	Structure	Weights	Total
NLTCS	7	151	157	2	1,087	1,089
MSNBC	155	626	782	126	10,471	10,597
KDDCup 2000	1,469	2,567	4036	780	22,421	23,201
Plants	126	6,348	6,474	34	16,084	16,118
Audio	138	8,473	8,611	49	28,688	28,738
Jester	79	23,690	23,769	27	13,745	13,771
Netflix	157	49,426	49,582	47	26,513	26,560
Accidents	213	41,769	41,982	43	25,393	25,436
Retail	306	10,392	10,698	112	24,918	25,030
Pumsb Star	293	66,679	66,971	43	17,534	17,578
DNA	38	4,532	4,570	6	655	661
Kosarak	1,210	20,143	21,353	238	47,125	47,363
MSWeb	2,254	28,140	30,393	485	46,878	47,363
Book	1,694	36,751	38,444	256	40,153	40,409
EachMovie	1,321	41,602	42,923	191	25,681	25,873
WebKB	2,052	60,959	63,010	212	21,901	22,113
Reuters-52	5,713	112,453	118,166	589	81,878	82,467
20 Newsgroups	9,971	354,541	364,512	1,455	234,323	235,778
BBC	1,555	54,738	56,293	173	8,488	8,661
Ad	4,510	18,865	23,375	740	14,634	15,373
Arith. Mean	1,663	47,142	48,805	280	35,428	35,709
Geom. Mean	507	17,060	18,267	110	18,987	19,173

Table 5: Run time for Ravikumar et al.'s algorithm and DTSL divided into time spent on structure learning and weight learning. Time is in seconds and includes parameter tuning.

require a decision tree with 2^k leaves, even though such interactions could be represented exactly by $O(k)$ features.

DT-BLM extends DTSL by further refining the features it generates, merging them into a smaller set of more essential features. This leads to modest but consistent gains in accuracy over DTSL at the cost of significantly longer learning times.

DT+L1 extends DTSL by including the features from L1, allowing it to capture many pairwise interactions and some higher-order interactions in the same model. As a result, DT+L1 does well overall across all datasets. DT+L1 is slower than DTSL but still much faster than DT-BLM. The accuracy of DT+L1 could perhaps be improved by performing additional tuning, rather than simply combining the models learned by DTSL and L1, or by using the features from DT-BLM. However, these modifications would also increase the learning time. The main risk of the expanded feature set used by DT+L1 is overfitting, which could explain its slightly worse performance on Jester and BBC.

8. Conclusions and Future Work

In this paper, we presented three new methods for using decision trees to learn the structure of Markov networks: DTSL, DT-BLM, and DT+L1.

DTSL is similar to the approach of Ravikumar et al. (2010), except that it uses decision trees in place of L1-regularized logistic regression. This allows it to learn longer features capturing interactions among more variables, which yields substantially better performance in several domains. DTSL is also similar to methods for learning dependency networks with tree conditional probability distributions (Heckerman et al., 2000). However, dependency networks may not represent consistent probability distributions and require that inference be done with Gibbs sampling, while the Markov networks learned by DTSL have neither of those limitations.

In terms of speed, we found that DTSL and L1-regularized logistic regression (Ravikumar et al., 2010) had similar speed, while BLM (Davis and Domingos, 2010) and Della Pietra et al. (1997) were significantly slower. With a faster weight learning method, this comparison would be even more favorable to DTSL and L1, since most of their time was spent on the final weight learning step. In terms of accuracy, DTSL is comparable in accuracy to other approaches, placing ahead of all three baselines on nine out of 20 datasets.

The other two methods are extensions of DTSL that combine it with other structure learning algorithms. DT-BLM builds on DTSL by running the BLM bottom-up structure learning algorithm on the features generated by DTSL. This usually leads to slightly better accuracy, but is also much slower. DT+L1 extends DTSL by adding the features learned by L1-regularized logistic regression. This hybrid approach is very effective: DT+L1 is one of the most accurate methods on 15 out of 20 datasets and runs much faster than DT-BLM.

Future work includes exploring other methods of learning local structure, such as rule sets, boosted decision trees, and neural networks; determining sufficient conditions for the asymptotic consistency of local learning; further improving speed, perhaps by using frequent itemsets; and incorporating faster methods for weight learning, since structure learning is no longer the bottleneck.

Acknowledgments

The authors would like to thank the anonymous reviewers for many helpful suggestions. We also thank Jan Van Haaren for his valuable feedback on the article. DL is partly supported by ARO grant W911NF-08-1-0242 and NSF grant IIS-1118050. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO or the United States Government. JD is partially supported by the research fund KU Leuven (CREA/11/015 and OT/11/051), and EU FP7 Marie Curie Career Integration Grant (#294068).

References

G. Andrew and J. Gao. Scalable training of l1-regularized log-linear models. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, pages 33–40. ACM

Press, 2007.

J. Besag. Statistical analysis of non-lattice data. *The Statistician*, 24:179–195, 1975.

C. Blake and C. J. Merz. UCI repository of machine learning databases. Machine-readable data repository, Department of Information and Computer Science, University of California at Irvine, Irvine, CA, 2000. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

F. Bromberg, D. Margaritis, and V. Honavar. Efficient Markov network structure discovery using independence tests. *Journal of Artificial Intelligence Research*, 35(2):449, 2009.

D. Chickering, D. Heckerman, and C. Meek. A Bayesian approach to learning Bayesian networks with local structure. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 80–89, Providence, RI, 1997. Morgan Kaufmann.

J. Davis and P. Domingos. Bottom-up learning of Markov network structure. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, pages 271–278, Haifa, Israel, 2010. ACM Press.

S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:380–392, 1997.

P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, Boston, MA, 2000. ACM Press.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, (9):1871–1874, 2008.

W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, London, UK, 1996.

K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

J. Van Haaren and J. Davis. Markov network structure learning: A randomized feature generation approach. In *Proceedings of the Twenty-Sixth National Conference on Artificial Intelligence*. AAAI Press, 2012.

D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, 2000.

G. Hulten and P. Domingos. Mining complex models from arbitrarily large databases in constant time. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 525–531, Edmonton, Canada, 2002. ACM Press.

R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. KDD-Cup 2000 organizers' report: Peeling the onion. *SIGKDD Explorations*, 2(2):86–98, 2000.

A. Kulesza and F. Pereira. Structured learning with approximate inference. In *Advances in Neural Information Processing Systems 20*, pages 785–792, 2007.

S.-I. Lee, V. Ganapathi, and D. Koller. Efficient structure learning of Markov networks using L1-regularization. In *Advances in Neural Information Processing Systems 19*, pages 817–824. MIT Press, 2007.

D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(3):503–528, 1989.

D. Lowd and J. Davis. Learning Markov network structure with decision trees. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM)*, pages 334–343, Sydney, Australia, 2010. IEEE Computer Society Press.

A. McCallum. Efficiently inducing features of conditional random fields. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 403–410, Acapulco, Mexico, 2003. Morgan Kaufmann.

K. Murphy, Y. Weiss, and M. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 467–475. Morgan Kaufmann, Stockholm, Sweden, 1999.

P. Ravikumar, M. J. Wainwright, and J. Lafferty. High-dimensional ising model selection using L1-regularized logistic regression. *Annals of Statistics*, 38(3):1287–1319, 2010.

M. Schmidt and K. Murphy. Convex structure learning in log-linear models: Beyond pairwise potentials. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. Springer, New York, NY, 1993.

I. Tsamardinos, L. Brown, and C. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.

C. Ziegler, S. McNee, J. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International World Wide Web Conference*, pages 22–32, 2005.